MVAPICH 0.9.9 User and Tuning Guide

MVAPICH Team

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University

http://mvapich.cse.ohio-state.edu/

Last Revised: June 14, 2007

Contents

1 Overview of the Open-Source MVAPICH Project

InfiniBand is emerging as a high-performance interconnect delivering low latency and high bandwidth. It is also getting widespread acceptance due to its *open standard*.

MVAPICH (pronounced as "em-vah-pich") is an *open-source* MPI software to exploit the novel features and mechanisms of InfiniBand and other RDMA enabled interconnects to deliver performance and scalability to MPI applications. This software is developed in the Network-Based Computing Laboratory (NBCL), headed by Prof. Dhabaleswar K. (DK) Panda.

Currently, there are two versions of this MPI: MVAPICH with MPI-1 semantics and MVAPICH2 with MPI-2 semantics. This *open-source* MPI software project started in 2001 and a first high-performance implementation was demonstrated at Supercomputing '02 conference. After that, this software has been steadily gaining acceptance in the HPC and InfiniBand community. As of the 06/14/2007, more than 525 organizations (National Labs, Universities, and Industry) in 30 countries have downloaded this software from OSU's web site directly. In addition, many IBA vendors, server vendors, and systems integrators have been incorporating MVAPICH/MVAPICH2 into their software stacks and distributing it. Several InfiniBand systems using MVAPICH have obtained positions in the TOP 500 ranking. The current version of MVAPICH is also being made available with the OpenFabrics/Gen2 stack. Both MVAPICH and MVAPICH2 distributions are available under BSD licensing.

More details on MVAPICH/MVAPICH2 software, users list, sample performance numbers on a wide range of platforms and interconnect, a set of OSU benchmarks, related publications, and other InfiniBand-related projects (parallel file systems, storage, data centers) can be obtained from the following URL:

http://mvapich.cse.ohio-state.edu/

This document contains necessary information for MVAPICH users to download, install, test, use, and tune MVAPICH 0.9.9. As we get feedback from users and take care of bug-fixes, we introduce new patches against our released distribution and also continuously update this document. Thus, we strongly request you to refer to our web page for updates.

2 How to use this User Guide?

This guide is designed to take the user through all the steps involved in configuring, installing, running and tuning MPI applications over InfiniBand using MVAPICH-0.9.9.

In Section 3 we describe all the features in MVAPICH 0.9.9. As you read through this section, please note our new features (highlighted as NEW). Some of these features are designed in order to optimize specific type of MPI applications and achieve greater scalability. Section 4 describes in detail the configuration and installation steps. This section enables the user to identify specific compilation flags which can be used to turn some of the features on of off. Usage instructions for MVAPICH are explained in Section 5. Apart from describing how to run simple MPI applications, this section also talks about running MVAPICH with some of the advanced features. Section 6 describes the usage of the OSU Benchmarks. If you have any problems using MVAPICH, please check Section 7 where we list some of the common problems users face. In Section 8 we suggest some tuning techniques for multi-thousand node clusters using some of our new features. Finally in Section 9, we list all important run-time and compile time parameters, their default values and a small description of what that parameter stands for.

3 MVAPICH 0.9.9 Features

MVAPICH (MPI-1 over InfiniBand) is an MPI-1 implementation based on MPICH and MVICH. MVAPICH 0.9.9 is available as a single integrated package (with the latest MPICH 1.2.7 and MVICH).

A complete set of features of MVAPICH 0.9.9 are:

- Single code base with multiple underlying transport interfaces
  - OpenFabrics
    - This interface support has the highest performing and most scalable features (such as On-demand connection management, SRQ, Shared Memory Collectives (NEW), RDMA-based collectives, multi-rail with advanced scheduling schemes, scalable MPD-based startup, etc.)
  - (NEW) Shared-Memory only channel
    - This interface support is useful for running MPI jobs on multi-processor systems without using any high-performance network. For example, multi-core servers, desktops, and laptops; and clusters with serial nodes.
  - uDAPL
    - The uDAPL interface to provide portability across networks and platforms with highest performance and scalability.
  - (NEW) QLogic InfiniPath
    - This interface provides native support for InfiniPath adapters from QLogic.
  - VAPI
  - TCP/IP
    - The standard TCP/IP interface (provided by MPICH) to work with a wide range of networks. This interface can also be used with IPoIB support of InfiniBand. However, it will not deliver good performance/scalability as compared to the other three interfaces.

- Designs for scaling to multi-thousand nodes with highest performance and reduced memory usage
  - (NEW) Message coalescing support to enable reduction of per Queue-pair send queues for reduction in memory requirement on large scale clusters. This design also increases the small message messaging rate significantly.
  - On-demand connection management using native InfiniBand Unreliable Datagram (UD) support. This feature enables InfiniBand connections to be setup dynamically, enhancing the scalability of MVAPICH on clusters of thousands of nodes.
  - Shared Receive Queue with Flow Control. The new design uses significantly less memory for MPI library.
  - Adaptive RDMA Fast Path
  - RDMA Polling Set
- Designs for avoiding hot-spots in networks of large-scale clusters
  - (NEW) Multi-pathing support for hot-spot avoidance in large scale clusters using LMC mechanism
  - (NEW) Multi-port/Multi-HCA support for enabling user processes to bind to different IB ports for balanced communication performance on multi-core platforms
- Scalable job startup using in-band IB communication using MPD
  - Flexibility for using rsh/ssh-based startup
- Optimized intra-node communication support by taking advantage of shared-memory communication
  - (NEW) Multi-core aware scalable shared memory design
  - Bus-based SMP systems
  - NUMA-based SMP systems
  - Processor Affinity
- Support for Fault Tolerance
  - Mem-to-mem reliable data transfer (detection of I/O bus error with 32bit CRC and retransmission in case of error) This mode enables MVAPICH to deliver messages reliably in presence of I/O bus errors.
  - Additional fault tolerance support (such as checkpoint restart, automatic path migration (APM), etc.) will be introduced in successive releases
- Single code base for following platforms (Architecture, OS, compilers, Devices, and InfiniBand adapters):

    - Architecture (tested with): EM64T, Opteron, IA-32 and IBM PPC
    - Operating Systems: Linux and Solaris
    - Compilers: gcc, intel, PathScale and PGI
    - Devices: VAPI, uDAPL, InfiniPath, OpenFabrics/Gen2, multi-rail and TCP/IP
    - InfiniBand adapters (tested with):
        * Mellanox adapters with PCI-X and PCI-Express (SDR and DDR with mem-full and mem-free cards)

- Optimized RDMA Write-based scheme for Eager protocol (short message transfer)
- Optimized implementation of Rendezvous protocol (large message transfer) for better computation-communication overlap and progress
    - RDMA Write-based
    - RDMA Read-based
- Two modes of communication progress
    - Polling
    - Blocking
- Advanced AVL tree-based resource-aware registration cache
    - (NEW) Memory Hook Support provided by integration with ptmalloc2 library. This provides safe release of memory to the Operating System and is expected to benefit the memory usage of applications that frequently use malloc and free operations.
- High performance and scalable collective communication support
    - (NEW) Optimized, high-performance shared memory aware collective operations for multi-core platforms
    - Broadcast support using IBA hardware multicast mechanism
    - RDMA-based Barrier support
    - RDMA-based All-to-all support
    - RDMA-based All-Gather support
    - Tuning and Optimization of various collective algorithms for a wide range of system sizes
- High performance and Portable Support for multiple networks through uDAPL interface. Tested with the following uDAPL libraries:
    - InfiniBand

    - uDAPL over OpenFabrics on Linux
    - uDAPL over IBTL on Solaris
  - Myrinet (DAPL-GM Beta)

  This uDAPL support is generic and can work with other networks that provide uDAPL interface. Please note that the stability and performance of MVAPICH with uDAPL depends on the stability and performance of the underlying uDAPL library being used.

- Schemes for minimizing memory resource usage on large scale systems
  - Automatic tuning for small, medium and large clusters
  - Shared Receive Queue support
  - On-Demand Connection management
- Multi-rail communication support
  - Multiple queue pairs per port
  - Multiple ports per adapter
  - Multiple adapters
  - Flexible scheduling policies
    - Separate control of small and large message scheduling
    - Three different scheduling policies for small messages: Using first sub channel, Round Robin and Process Binding
    - Six different scheduling policies for large messages: Round Robin, Weighted striping, Even striping, Stripe Blocking, Adaptive Striping and Process Binding.
- Shared library support for existing binary MPI application programs to run
- Shared library support for Solaris
- ROMIO support
- Support for TotalView debugger
- Integrated and easy-to-use build script which automatically detects system architecture and InfiniBand adapter types and optimizes MVAPICH for any particular installation
- Tuned thresholds and associated optimizations for
  - different architectures/platforms mentioned above
  - different memory/system bus characteristics

  - different network interfaces (PCI-X, PCI-Express with SDR and DDR and IBM ehca adapter with GX interface)
  - different networks enabled by multiple devices/interfaces

- Incorporates a set of runtime and compile time tunable parameters (at MPI and network layers) for convenient tuning on
  - large scale systems
  - future platforms

The MVAPICH 0.9.9 package and the project also includes the following provisions:

- Public SVN access of the code base
- A set of micro-benchmarks for carrying out MPI-level performance evaluation after the installation
- Public mvapich-discuss mailing list for mvapich users to
  - ask for help and support from each other and get prompt response
  - enable users and developers to contribute patches and enhancements

4 Installation Instructions

4.1 Download MVAPICH source code

The MVAPICH 0.9.9 source code package includes the latest MPICH 1.2.7 version and also the required MVICH files from LBNL. Thus, there is no need to download any other files except MVAPICH 0.9.9 source code.

You can go to the MVAPICH website to obtain the source code.

4.2 Prepare MVAPICH source code

Untar the archive you have downloaded from the web page using the following command. You will have a directory named `mvapich-0.9.9` after executing this command.

```
$ tar xzf mvapich-0.9.9.tar.gz
```

4.3 Getting MVAPICH source updates

As we enhance and improve MVAPICH, we update the available source code on our public SVN repository. In order to obtain these updates, please install a SVN client on your machine. The latest MVAPICH sources may be obtained from the "trunk" of the SVN using the following command:

```
$ svn co https://mvapich.cse.ohio-state.edu/svn/mpi/mvapich/trunk
```

The "trunk" may contain newer features and bug fixes. However, it is likely to be lightly tested. If you are interested in obtaining stable and major bug fixes to any release version, you should update your sources from the "branch" of the SVN using the following command:

```
$ svn co https://mvapich.cse.ohio-state.edu/svn/mpi/mvapich/branches/0.9.9
```

MVAPICH 0.9.9 provides support for seven different ADI devices. Namely, Gen2 Single-Rail (`ch_gen2`), Gen2 Multi-Rail (`ch_gen2_multirail`), Shared memory device (`ch_smp`), VAPI Single-Rail (`vapi`), VAPI Multi-Rail (`vapi_multirail`), uDAPL (`udapl`) and QLogic InfiniPath (`psm`). Additionally, you can also configure MVAPICH over the standard TCP/IP interface and use it over IPoIB.

In the following section we describe how to build and configure the Single-Rail device. In later sections 4.4.5, 4.4.6, 4.4.3, 4.4.7 and 4.4.8, we describe the building and configuration of the Multi-Rail, uDAPL, InfiniPath, Shared memory device and TCP devices, respectively.

4.4 Build MVAPICH

There are several options to build MVAPICH 0.9.9 based on the underlying InfiniBand libraries you want to utilize. In this section we describe in detail the steps you need to perform to correctly build MVAPICH on your choice of InfiniBand libraries, namely OpenFabrics Gen2, Mellanox VAPI, uDAPL or QLogic InfiniPath.

4.4.1 Build MVAPICH with Single-Rail configuration on OpenFabrics Gen2

There are several methods to configure MVAPICH 0.9.9.

- **Using Default Configuration:** Go to the `mvapich-0.9.9` directory. We have included a single script for OpenFabrics/Gen2 (`make.mvapich.gen2`) that takes care of different platforms, compilers and architectures. By default, the compilation script uses `gcc`. In order to select your compiler, please set the variable `CC` in the script to use either Intel, PathScale or PGI compiler. The platform/architecture is detected automatically.
- **Customize MVAPICH Configuration:** MVAPICH has many optimization schemes to enhance performance. While some schemes can be turned on/off by the compilation flags which are listed in the variable `CFLAGS` (in the default compilation script), many more optimizations and tuning is possible using environmental parameters. A list of all possible parameters is in Section 9.
  - Ring Based QP exchange: Efficient job startup using ring-based QP information exchange. Controlled by `-DUSE_MPD_RING`.
  - Memory-to-memory Reliability: When compiled with this support, MVAPICH performs memory-to-memory error checking for each data-transfer between a pair of nodes. This is useful for detecting errors across I/O buses. In case any error is detected during the transfer, MVAPICH will re-transmit corrupted messages. Controlled by `-DMEMORY_RELIABLE`.
  - InfiniBand Hardware Multicast: This utilizes InfiniBand hardware multicast feature to efficiently implement the MPI broadcast collective. Controlled by `-DMCST_SUPPORT` and `-DOSM_VENDOR_INTF_TS`. Please note this feature is not fully enabled (for OpenFabrics/Gen2 stack) even though the user code is present in MVAPICH 0.9.9. This is because the since the OpenFabrics community hasn't yet finalized the interface for hardware multicast. This feature is available with VAPI single-rail and VAPI multi-rail devices.
  - Shared library support: Enables the use of shared libraries. The flag `--enable-sharedlib` should be added as a parameter to `configure` in the default `make.mvapich.gen2` script.

  - TotalView Debugger support: This enables the use of TotalView debugger for your MPI applications. In order to enable this feature, you need to pass `--enable-sharedlib` and `--enable-debug` options to `configure` in the `make.mvapich.gen2` script. Please note that currently `mpirun_rsh` is required to run applications with TotalView support.

After setting all the parameters, the script `make.mvapich.gen2` configures, builds and installs the entire package in the directory specified by the variable `PREFIX`.

4.4.2 Build MVAPICH with Multi-Rail Configuration on OpenFabrics Gen2

There are several methods to configure MVAPICH 0.9.9 with multi-rail device on OpenFabrics Gen2.

- **Using Default Configuration:** Go to the `mvapich-0.9.9` directory. We have included a single script for Gen2 (`make.mvapich.gen2_multirail`) that takes care of different platforms, compilers and architectures. By default, the compilation script uses `gcc`. In order to select your compiler, please set the variable `CC` in the script to use either Intel, PathScale or PGI compilers. The platform/architecture is detected automatically.
- **Customize MVAPICH Configuration:** MVAPICH has many optimization schemes to enhance performance. These schemes can be turned on/off by the compilation flags which are listed in the variable `CFLAGS` (in the default compilation script). Most of these options are the same as described in section 4.4.5. The only exception being that `-DUSE_MPD_RING` is not yet supported for this device.

After setting all the parameters, the script `make.mvapich.gen2_multirail` configures, builds and installs the entire package in the directory specified by the variable `PREFIX`.

MVAPICH provides multiple scheduling policies for communication, in the presence of multiple ports/adapters/paths with the multi-rail configuration. Please refer to 5.8 for more details.

4.4.3 Build MVAPICH with QLogic InfiniPath

There are several methods to configure MVAPICH 0.9.9.

- **Using Default Configuration:** Go to the `mvapich-0.9.9` directory. We have included a single script for InfiniPath (`make.mvapich.psm`) that takes care of different platforms, compilers and architectures. By default, the compilation script uses `gcc`. In order to select your compiler, please set the variable `CC` in the script to use either Intel, PathScale or PGI compiler. The platform/architecture is detected automatically.

- **Customize MVAPICH Configuration:** MVAPICH has many optimization schemes to enhance performance. While some schemes can be turned on/off by the compilation flags which are listed in the variable `CFLAGS` (in the default compilation script), many more optimizations and tuning is possible using environmental parameters. A list of all possible parameters is in Section 9.
  - TotalView Debugger support: This enables the use of TotalView debugger for your MPI applications. In order to enable this feature, you need to pass `--enable-sharedlib` and `--enable-debug` options to `configure` in the `make.mvapich.psm` script. Please note that currently `mpirun_rsh` is required to run applications with TotalView support.

After setting all the parameters, the script `make.mvapich.psm` configures, builds and installs the entire package in the directory specified by the variable `PREFIX`.

4.4.4 Build MVAPICH with Single-Rail Configuration on VAPI

There are several methods to configure MVAPICH 0.9.9 on VAPI.

- **Using Default Configuration:** Go to the `mvapich-0.9.9` directory. We have included a single script for VAPI (`make.mvapich.vapi`) that takes care of different platforms, compilers and architectures. By default, the compilation script uses `gcc`. In order to select your compiler, please set the variable `CC` in the script to use either Intel, PathScale or PGI compilers. The platform/architecture is detected automatically. The script prompts the user for parameters like I/O Bus type (`PCI-X`, `PCI-Ex`) and InfiniBand link speed (`SDR`, `DDR`).
- **Customize MVAPICH Configuration:** The following customizations are available for the `vapi` device.
  - Ring Based QP exchange: Efficient job startup using ring-based QP information exchange. Controlled by `-DUSE_MPD_RING`.
  - RDMA Read: This allows MVAPICH to achieve better overlap of communication and computation. Controlled by `-DVIADEV_RGET_SUPPORT`. Please note that if you choose this option, you will have to disable (ie. remove from CFLAGS variable in the script), `-DVIADEV_RPUT_SUPPORT`.
  - Blocking progress: This allows MVAPICH to yield CPU resources to other processes when waiting for incoming messages from the network. Controlled by `-DBLOCKING_SUPPORT`. Please note that if you use this option, you will have to disable (ie. remove from CFLAGS variable in the script), `-DADAPTIVE_RDMA_FAST_PATH`, `-DRDMA_FAST_PATH`, `-D_SMP_` and `-D_SMP_RNDV_`.

  - InfiniBand Hardware Multicast: This utilizes InfiniBand hardware multicast feature to efficiently implement the MPI broadcast collective. Controlled by `-DMCST_SUPPORT` and `-DOSM_VENDOR_INTF_TS`.
  - Shared library support: Enables the use of shared libraries. The flag `--enable-sharedlib` should be added as a parameter to configure in the default `make.mvapich.vapi` script.
  - TotalView Debugger support: This enables the use of TotalView debugger for your MPI applications. In order to enable this feature, you need to pass `--enable-sharedlib` and `--enable-debug` options to configure in the `make.mvapich.vapi` script. Please note that currently `mpirun_rsh` is required to run applications with TotalView support.

After setting all the parameters, the script `make.mvapich.vapi` configures, builds and installs the entire package in the directory specified by the variable `PREFIX`.

4.4.5 Build MVAPICH with Multi-Rail Configuration on VAPI

There are several methods to configure MVAPICH 0.9.9 with multi-rail device.

- **Using Default Configuration:** Go to the `mvapich-0.9.9` directory. We have included a single script for VAPI (`make.mvapich.vapi_multirail`) that takes care of different platforms, compilers and architectures. By default, the compilation script uses `gcc`. In order to select your compiler, please set the variable `CC` in the script to use either Intel, PathScale or PGI compilers. The platform/architecture is detected automatically.
- **Customize MVAPICH Configuration:** MVAPICH has many optimization schemes to enhance performance. These schemes can be turned on/off by the compilation flags which are listed in the variable `CFLAGS` (in the default compilation script).
  - Ring Based QP exchange: Efficient job startup using ring-based QP information exchange. Controlled by `-DUSE_MPD_RING`.
  - Shared memory communication: Intra-node communication which uses shared memory instead of HCA loop back. Controlled by `-D_SMP_` and `-D_SMP_RNDV_`.
  - InfiniBand Hardware Multicast: This utilizes InfiniBand hardware multicast feature to efficiently implement the MPI broadcast collective. Controlled by `-DMCST_SUPPORT` and `-DOSM_VENDOR_INTF_TS`. Please note that you should have corresponding OpenSM libraries installed to utilize this feature.
  - Shared library support: Enables the use of shared libraries. The flag `--enable-sharedlib` should be added as a parameter to `configure` in the default `make.mvapich.vapi_multirail` script.

After setting all the parameters, the script `make.mvapich.vapi_multirail` configures, builds and installs the entire package in the directory specified by the variable `PREFIX`.

4.4.6 Build MVAPICH with Single-Rail Configuration on uDAPL

Before installing MVAPICH with uDAPL, please make sure you have the uDAPL library installed properly.

There are several methods to configure MVAPICH 0.9.9 with uDAPL.

- **Using Default Configuration:** Go to the `mvapich-0.9.9` directory. We have included a single script for uDAPL (`make.mvapich.udapl`) that takes care of different platforms, compilers and architectures. By default, the compilation script uses `GNU compilers`. In order to select different compilers, please set the variable `CC`, in the script. The platform/architecture is detected automatically. Because there are many varieties of uDAPL packages, the script `make.mvapich.udapl` requires two parameters: *dat_library_path* and *dat_include_path*, before it can start building the package. These are the library path and include path of your DAPL installation, respectively. In addition, the script prompts the user for parameters like cluster size.
- **Customize MVAPICH configuration:** MVAPICH has many optimization schemes to enhance performance. These schemes can be turned on/off by the compilation flags which are listed in the variable `CFLAGS` (in the default compilation script).
  - Shared memory communication: Intra-node communication which uses shared memory instead of NIC loop back. Controlled by `-D_SMP_` and `-D_SMP_RNDV_`.
  - Shared library support: Enables the use of shared libraries. The flag `--enable-sharedlib` should be added as a parameter to `configure` in the default `make.mvapich.udapl` script.
  - TotalView Debugger support: This enables the use of TotalView debugger for your MPI applications. In order to enable this feature, you need to pass `--enable-sharedlib` and `--enable-debug` options to `configure` in the `make.mvapich.udapl` script. Please note that currently `mpirun_rsh` is required to run applications with TotalView support.

After setting all the parameters, the script `make.mvapich.udapl` configures, builds and installs the entire package in the directory specified by the variable `PREFIX`.

4.4.7 Build MVAPICH with Shared Memory Device

In the `mvapich-0.9.9` directory, we have provided a script `make.mvapich.smp` for building MVAPICH over shared memory intended for single SMP systems. The script `make.mvapich.smp` takes care of different platforms, compilers and architectures. By default, the compilation script uses `gcc`. In order to select your compiler, please set the variable `CC` in the script to use either Intel, PathScale or PGI compilers. The platform/architecture is detected automatically. The usage of the shared memory device can be found in 5.2.

4.4.8 Build MVAPICH with TCP/IPoIB

In the `mvapich-0.9.9` directory, we have provided a script `make.mvapich.tcp` for building MVAPICH over TCP/IP intended for use over IPoIB (IP over InfiniBand). In order to select any other compiler than `GCC`, please set your `CC` variable in that script. Simply execute this script (e.g. `./make.mvapich.tcp`) for completing your build.

5 Usage Instructions

This section discusses the usage methods for the various features provided by MVAPICH. If you face any problem while following these instructions, please refer to Section 7.

5.1 Compile MPI applications

Use `mpicc, mpif77, mpiCC`, or `mpif90` to compile applications. They can be found under `mvapich-0.9.9/bin`.

There are several options to run MPI applications. Please select one of the following options based on your need.

5.2 Run MPI applications using `mpirun_rsh`

Prerequisites:

- Either `ssh` or `rsh` should be enabled between the front nodes and the computing nodes. In addition to this setup, you should be able to login to the remote nodes without any password prompts.
- Configuring and installing MVAPICH without MPD support.

Examples of running programs using `mpirun_rsh`:

```
$ mpirun_rsh -np 4 n0 n1 n2 n3 ./cpi
```

The above command runs `cpi` on nodes n0, n1, n2 and n3 nodes, one process per each node. By default `ssh` is used.

```
$ mpirun_rsh -rsh -np 4 n0 n1 n2 n3 ./cpi
```

The above command runs `cpi` on nodes n0, n1, n2 and n3 nodes, one process per each node. `rsh` is used regardless of whether `ssh` or `rsh` is used when compiling MVAPICH.

```
$ mpirun_rsh -np 4 -hostfile hosts ./cpi
```

A list of nodes are in hosts, one per line. MPI ranks are assigned in order of the hosts listed in the hosts file or in the order they are passed to mpirun_rsh. ie. if the nodes are listed as n0 n1 n0 n1, then n0 will have two processes, rank 0 and rank 2; whereas n1 will have rank 1 and 3. This rank distribution is known as “cyclic”. If the nodes are listed as n0 n0 n1 n1, then n0 will have ranks 0 and 1; whereas n1 will have ranks 2 and 3. This rank distribution is known as “block”.

If you are using the shared memory device, then host names can be omitted:

```
$ mpirun_rsh -np 4 ./cpi
```

Many parameters of the MPI library can be very easily configured during run-time using environmental variables. In order to pass any environment variable to the application, simply put the variable names and values just before the executable name, like in the following example:

```
$ mpirun_rsh -np 4 -hostfile hosts ENV1=value ENV2=value ./cpi
```

Note that the environmental variables should be put immediately before the executable.

Alternatively, you may also place environmental variables in your shell environment (e.g. `.bashrc`). These will be automatically picked up when the application starts executing.

Please note that there are many different parameters which could be used to improve the performance of applications depending upon their requirements from the MPI library. For a discussion on how to identify which variables may be of interest to you, please take a look at Section 8.

Other options of `mpirun_rsh` can be obtained using

```
$ mpirun_rsh --help
```

5.3 Run MPI applications using MPD

MVAPICH is provided with MPD support for fast process startup. Be sure to do `make install` to have MPD system installed into the correct directory. This is a general requirement for using extended features, such as MPD and TotalView, with MPICH.

To know more about MPD, please refer to the MPD documents provided along Argonne MPICH release. This should be available as `mvapich-0.9.9/doc/mpichman-chp4mpd.pdf`, section 4.9. An online document is also available from MPICH website.

Step by step instructions to setup MPD environment manually:

- First log into node00 and then proceed with the following steps:
  - Be sure you have `.mpd.conf` and `.mpdpasswd` in your home directory. They can be a single line file like the following:
    `secretword=56rtG9`
  - Include MPD path into your path
    ```
    $ export MPD_BIN=$MVAPICH_HOME/bin PATH=$MVAPICH_HOME/bin:$PATH
    ```
    `$MVAPICH_HOME` is the installation path of your MVAPICH, as specified by `--prefix` when you configure MVAPICH.
  - run `mpd` on node00

- Find out the port number this daemon is exposing, i.e. typically a number from the following trace command. In the following lines this number is assumed to be 33333.
  ```
  $ mpdtrace
  ```
- Launch another daemon on node01
  ```
  $ ssh -n node01 ${MPD_BIN} -h node00 -p 33333 &
  ```
- Simple Testing
  ```
  $ mpirun_mpd -np 4 cpi
  ```
- Cleanup
  ```
  $ mpdallexit
  ```

• We provide a script in the `mvapich-0.9.9` directory. You can use this script to expedite MPD setup.

- Be sure you have .mpd.conf and .mpdpasswd in your home directory. They can be a single line file like the following:
  password=56rtG9
- Make a sample machine file, hostfile, which reads
  node00
  node01
  node02
  node03
- Startup daemons:
  ```
  $ mvapich.mpd.sh start hostfile $MPD_BIN
  ```
- Stop daemons:
  ```
  $ mvapich.mpd.sh stop hostfile $MPD_BIN
  ```
- Cleanup daemons if you have trouble:
  ```
  $ mvapich.mpd.sh cleanup hostfile $MPD_BIN
  ```

• Environmental variables setup using mpirun_mpd in MVAPICH
```
$ mpirun_mpd -np $np $prog <args> -MPDENV-
ENV1=value1 ENV2=value2
```
Details can be referred from MPICH Website.

5.4 Run MPI applications with Shared Memory Collectives

MVAPICH provides shared memory implementations of three important collectives: `MPI_Allreduce`, `MPI_Reduce` and `MPI_Barrier`. These collective operations are enabled by default. Shared Memory Collectives are supported over Gen2 and Shared Memory devices.

These operations can be disabled all at once by setting VIADEV_USE_SHMEM_COLL to 0 or one at a time by using the following environment variables:

- To disable Shmem MPI_Allreduce: VIADEV_USE_SHMEM_ALLREDUCE=0
- To disable Shmem MPI_Reduce: VIADEV_USE_SHMEM_REDUCE=0
- To disable Shmem MPI_Barrier: VIADEV_USE_SHMEM_BARRIER=0

Please refer to section 9.7 for tuning the various environment variables.

5.5 Run MPI applications using shared library support

MVAPICH provides shared library support. This feature allows you to build your application on top of MPI shared library. If you choose this option, you still will be able to compile applications with static libraries. But as default, when you have shared library support enabled, your applications will be built on top of shared libraries automatically. The following commands provide some examples of how to build and run your application with shared library support.

- To compile your application with shared library support. Run the following command.

  ```
  $ mpicc -o cpi cpi.c
  ```

- To execute an application compiled with shared library support, you need to specify the path to the shared library by setting
  `LD_LIBRARY_PATH=<path-to-shared-libraries>` in the command line.

  For example,

  ```
  $ mpirun_rsh -np 2 n0 n1 LD_LIBRARY_PATH=$MVAPICH_BUILD/lib/shared
  ./cpi
  ```

  Again, note that "LD_LIBRARY_PATH=path-to-shared-libraries" should be put immediately before the executable file.

- To disable MVAPICH shared library support even if you have installed MVAPICH. Run the following command.

  ```
  $ mpicc -noshlib -o cpi cpi.c
  ```

5.6 Run MPI applications using TotalView Debugger support

MVAPICH provides TotalView support for the single-rail VAPI, InfiniPath and OpenFabrics/Gen2 devices, namely: `mpid/vapi`, `mpid/psm` and `mpid/ch_gen2`. You need to use `mpirun_rsh` when running TotalView. The following commands also provide an example of how to build and run your application with TotalView support. *Note: running TotalView demands correct setup in your environment, if you encounter any problem with your setup, please check with your system administrator for help.*

- Define ssh as a `TVDSVRLAUNCHCMD` variable in your default shell. For example, with bashrc, you can do
  ```
  $ echo "export TVDSVRLAUNCHCMD=ssh" >>  /.bashrc
  ```
- Configure MVAPICH with the configure options `--enable-debug --enable-sharedlib` in addition to the default options and then build MVAPICH.
- Compile your program with a flag -g
  ```
  $ mpicc -g -o prog prog.c
  ```
- Define the correct path to TotalView as the TOTALVIEW variable. For example, under bash shell:
  ```
  $ export TOTALVIEW=<path_to_TotalView>
  ```
- Run your program:
  ```
  $ mpirun_rsh -tv -np 2 n0 n1
  LD_LIBRARY_PATH=$MVAPICH_BUILD/lib/shared:$MVAPICH_BUILD/lib prog
  ```
- Trouble shooting:
  - X authentication errors: check if you have enabled X Forwarding
    `$ cat ''ForwardX11 yes'' >> $HOME/.ssh/config`
  - rsh connection time out: check if you have defined `TVDSVRLAUNCHCMD` as ssh in your default shell file, .bashrc, .cshrc, or the like.
  - ssh authentication error: ssh to the computer node with its long form host name, for example, ssh i0.domain.osu.edu

5.7 Run MPI applications with Multi-Pathing Support for Multi-Core Architectures

MVAPICH provides multi-rail device with advance scheduling policies for data transfer 5.8. However, even with the single-rail configuration, multi-pathing (multiple ports, adapters and multiple paths provided by the LMC mechanism) can be used for multi-core systems. With this support, processes executing on the same node can leverage the above configurations by binding to one of the available configuration. MVAPICH provides multiple choices to the user for leveraging this functionality, which are described in the upcoming examples. This functionality is currently available only in the single-rail gen2 device.

- To allow processes on the same node to use multiple ports in a round robin fashion
  ```
  $ mpirun_rsh -np 4 n0 n0 n1 n1 VIADEV_USE_MULTIPORT=1 ./cpi
  ```
- To allow processes on the same node to use multiple adapters in a round robin fashion
  ```
  $ mpirun_rsh -np 4 n0 n0 n1 n1 VIADEV_USE_MULTIHCA=1 ./cpi
  ```
- To allow processes on the same node to use multiple adapters and multiple ports in a round robin fashion
  ```
  $ mpirun_rsh -np 4 n0 n0 n1 n1 VIADEV_USE_MULTIHCA=1
  VIADEV_USE_MULTIPORT=1 ./cpi
  ```
  The usage of multiple paths is disabled by default. It's usage can be controlled by using the parameter VIADEV_USE_LMC (9.1.6).
- In the above examples, the binding of paths to processes is done in a round robin fashion. In addition, MVAPICH allows a user to explicitly specify the Adapter and Port to be used by the library. This can be done by specification in the hostfile as follows.
  ```
  $ cat hosts
  n0:mthca0:1
  n0:mthca1:2
  n1:mthca0:2
  n1:mthca1:1
  ```
  With this specification, process 0 would be bound to port1 of adapter "mthca0", process 1 to port 2 of adapter "mthca1" and so on.

5.8 Run MPI applications on Multi-Rail Configurations

MVAPICH provides multiple scheduling policies for communication, in the presence of multiple ports/adapters/paths with the multi-rail configuration. Run-time parameters are being provided to control the policies. They are further divided into policies for small and large messages. These policies are available in the multirail devices for gen2 and VAPI.

- **Scheduling Policies for Small Messages:** For this section, we refer to the messages below the rendezvous threshold as small messages. MVAPICH allows a run-time parameter `SM_SCHEDULING`, which schedules small messages depending upon the user input.
  - In this context, `USE_FIRST` policy forces all small messages to use only first path for communication.
  - The `ROUND_ROBIN` scheduling policy schedules the messages in a round robin fashion on available paths. This policy is helpful, in the presence of multiple independent paths, benefitting the throughput. As an example, in the presence of

two HCAs (1 port per HCA), HCA0 and HCA1 on a system, the messages of each process on the system will be sent through HCA0 and HCA1 in round robin fashion.

  – `PROCESS_BINDING` policy forces the processes on the same node to use the available paths on the node to be used in a round robin fashion. From the above example, each process on a system is bound to HCA0 or HCA1. Unlike ROUND_ROBIN policy, each process will be able to use only one HCA. Such policy is beneficial in providing fairer distribution of network paths to processes on the same node, particularly for the upcoming multi-way SMP and CMP systems.

- **Scheduling Policies for Large Messages:** In this section, we discuss the scheduling policies for messages of size greater than or equal to the rendezvous threshold. In addition to the policies discussed above, MVAPICH provides three scheduling policies, which can be specified by using `LM_SCHEDULING` runtime variable.

  – `EVEN_STRIPING` scheduling policy is useful in the presence of paths with equal available bandwidths.

  – In the presence of paths with different bandwidths, the `ADAPTIVE_STRIPING` policy can be used. Under this policy, the available bandwidths for each path are determined using a feedback loop and the message is striped accordingly, so that the load on each path is balanced.

  – The `STRIPE_BLOCKING` scheduling policy differentiates between the blocking and non-blocking communication at the ADI layer. This policy is useful with increasing number of paths, where the above-mentioned STRIPING policies may lead to high overhead due to multiple fragments created by the policy.

5.9 Run MPI applications using InfiniBand hardware Multicast based MPI Broadcast support

In MVAPICH 0.9.9, we provide a hardware multicast-based MPI Broadcast. Currently, this feature is supported over VAPI device. This will be supported over Gen2 from the following releases.

Prerequisites for using this support are:

- Subnet Management (SM) Support: We have developed and tested this feature using OpenSM. Thus, we provide instructions w.r.t. OpenSM. If you are using any other SM, please make appropriate adjustment to the following paths and steps.

- OpenSM has to be running continuously on one node for hardware multicast to work. We recommend to use a non-compute node as the node running Opensm. If OpenSM is not already running on one of the nodes connected to the subnet then follow the procedure below to run OpenSM.

  – Run opensm with a GUID choice. The GUID choice is essentially the port number. You can choose 1 or 2, which is the port you want to use. But make sure that whichever port you choose, it must be connected to the IB subnet.

5.9.1 Usage examples:

When MVAPICH is configured and installed with hardware multicast-based MPI Broadcast support, MPI_Bcast takes advantage of hardware multicast for broadcasting messages reliably.

This feature can be disabled by using an environment variable, `DISABLE_HARDWARE_MCST`, as shown below:

```
$ mpirun_rsh -np 4 n0 n1 n2 n3 DISABLE_HARDWARE_MCST=1 ./cpi
```

MPI_Bcast will use the original point-to-point based implementation in MPICH-1.2.7 when `DISABLE_HARDWARE_MCST` is set. Note that, `DISABLE_HARDWARE_MCST=1` should be put immediately before the executable file.

Important notes:

- If the multicast group create/join fails, restarting Opensm helps.
- If you are still facing the problem above, please create the multicast group manually by executing the `ibmcgrp` command found in the `$MVAPICH_HOME/bin` directory.

  example:

  ```
  $ ibmcgrp -c -g 0xff12a01cfe800000:HHHHHHHHHHHHHHHH
  --port_num 1
  ```
- In the current implementation we support a single multicast group which includes all the nodes. Thus MPI_COMM_WORLD and any communicator which includes all the nodes can take advantage of the hardware multicast based MPI_Bcast. We are working on extending this feature to support arbitrary communicators.

6 Using OSU Benchmarks

If you have arrived at this point, you have successfully installed MVAPICH. Congratulations!! In the `mvapich-0.9.9/osu_benchmarks` directory, we provide four basic performance tests: one-way latency test, uni-directional bandwidth test, bi-directional bandwidth test multiple bandwidth/message rate, and MPI-level broadcast latency test. You can compile and run these tests on your machines to evaluate the basic performance of MVAPICH.

These benchmarks as well as other benchmarks (such as for one-sided operations in MPI-2) are available on our projects' web page. Sample performance numbers for these benchmarks on representative platforms and IBA gears are also included on our projects' web page. You are welcome to compare your performance numbers with our numbers. If you see any big discrepancy, please let the MVAPICH community know by sending an email to the mailing list mvapich-discuss@cse.ohio-state.edu.

7 Troubleshooting

Based on our experience and feedback we have received from our users, here we include some of the problems a user may experience and the steps to resolve them. If you are experiencing any other problem, please feel free to contact the MVAPICH community by sending an email to the mailing list mvapich-discuss@cse.ohio-state.edu.

MVAPICH can be used over four underlying InfiniBand libraries, namely OpenFabrics (Gen2), VAPI, UDAPL and QLogic InfiniPath. Based on the underlying library being utilized, the troubleshooting steps may be different. However, some of the troubleshooting hints are common for all underlying libraries. Thus, in this section, we have divided the troubleshooting tips into four sections: General troubleshooting and Troubleshooting over any one of the three InfiniBand libraries.

7.1 General Troubleshooting

7.1.1 My application cannot pass `MPI_Init`

This is a common symptom of several setup issues related to job startup. Please make sure of the following things:

- If you have enabled `ssh` based startup, make sure that you have set up ssh keys for logging into all the nodes without any password prompt.
- If you have enabled `rsh` based startup, make sure that `rsh, rlogin` etc. are active on all the nodes and you can log in without any password prompts.
- Please make sure the host names supplied to MVAPICH for the particular job match the host names in file `/etc/hosts` present on each of the target nodes.
- Please make sure you can run some InfiniBand level program on the nodes you are trying to run MPI programs. Usually running `perf_main` (for VAPI), `ibv_rc_pingpong` (for OpenFabrics Gen2) or `dapltest` (for UDAPL) is a good choice.

7.1.2 My application hangs/aborts in `MPI_Alltoall`

MVAPICH implements highly optimized RDMA collective algorithms for frequently used collectives such as `MPI_Alltoall`, `MPI_Allgather`, `MPI_Barrier`. The optimized implementations have been well tested and tuned. However, if you face any problems in these collectives for your application, please disable the optimized collectives. You can do so by using:

```
$ mpirun_rsh -np 8 -hostfile hf DISABLE_RDMA_ALLTOALL=1 ./a.out
```

Similarly, you can use DISABLE_RDMA_ALLGATHER and DISABLE_RDMA_BARRIER to disable these features.

If your application fails for the default configuration (RDMA collectives enabled) and passes using the above command (after disabling an RDMA collective), please report the error to the MVAPICH community by sending an email to the list mvapich-discuss@cse.ohio-state.edu.

7.1.3 Cannot find mpd.conf

If you get this error, please set your .mpd.conf and .mpdpasswd files as mentioned in Section 5.3.

7.1.4 Building MVAPICH hangs with hardware multicast enabled

We have found out that on some of our machines, when we build MVAPICH with hardware multicast-based Broadcast support, the system may hang in the *make* step. If this happens, please provide "--disable-cxx" in your configure command or add this option in the configure command in the scripts provided with the package.

7.1.5 Building MVAPICH with g77/gfortran

The gfortran compiler can be used for F77 and F90. In order to make this work, the following environment variables should be set prior to running the build script:

```
$ export F77=gfortran
$ export F90=gfortran
$ export F77_GETARGDECL=" "
```

If g77 and gfortran are used together for F77 and F90 respectively, it might be necessary to set the following environment variable in order to get around possible compatibility issues:

```
$ export F90FLAGS="-ff2c"
```

7.1.6 Running MPI programs built with gfortran

MPI programs built with gfortran might not appear to run correctly due to the default output buffering used by gfortran. If it seems there is an issue with program output, the GFORTRAN_UNBUFFERED_ALL variable can be set to "y" when using mpirun_rsh to fix the problem. Running the pi3f90 example program using this variable setting is shown below:

```
$ mpirun_rsh -np 2 n1 n2 GFORTRAN_UNBUFFERED_ALL=y
./pi3f90
```

7.1.7 Other MPICH problems

Several well-known MPICH related problems on different platforms and environments have already been identified by Argonne. They are available on the MPICH patch webpage.

7.2 Troubleshooting with MVAPICH/OpenFabrics(Gen2)

In this section, we discuss the general error conditions for MVAPICH based on OpenFabrics Gen2.

7.2.1 No IB Devices found

This error is generated by MVAPICH when it cannot find any Gen2 InfiniBand devices. If you are experiencing this error, then please make sure that your Gen2 installation is proper. You can do so by doing the following:

```
$ locate libibverbs
```

This tells you if you have installed `libibverbs` (the Gen2 verbs layer) or not. By default it installs in `/usr/local`.

If you have installed `libibverbs`, then please check if the OpenFabrics Gen2 drivers are loaded. You can do so by:

```
$ lsmod | grep ib
```

If this command does not list `ib_uverbs`, then probably you haven't started all OpenFabrics Gen2 services. Please refer to the OpenFabrics Wiki installation cheat sheet for more details on setting up the OpenFabrics Gen2 stack.

7.2.2 Error getting HCA Context

This error is generated when MVAPICH cannot "open" the HCA (or the InfiniBand communication device). Please execute:

```
$ ls -l /dev/infiniband
```

If this command shows any devices `uverbs0` with read/write permissions for users as shown below, please consult the "Loading kernel components" section of the OpenFabrics Wiki installation cheat sheet.

```
crw-rw-rw- 1 root root 231, 192 Feb 24 14:31 uverbs0
```

7.2.3 CQ or QP Creation failure

If you encounter this error, then you need to set the maximum available locked memory value for your system. The usual Linux defaults are quite low to what is required for HPC applications. One way to do this is to edit the file `/etc/security/limits.conf` and enter the following line:

```
* soft memlock phys_mem_size_in_KB
```

Where, `phys_mem_size_in_KB` is the `MemTotal` value reported by `/proc/meminfo`. In addition, you need to enter the following line in `/etc/init.d/sshd` and then restart sshd.

```
ulimit -l phys_mem_size_in_KB
```

7.2.4 No Active Port found

MVAPICH generates this error when it cannot find any port active for the specific HCA being used for communication. This probably means that the ports are not configured to be a part of the InfiniBand subnet and thus are not "Active". You can check whether the port is active or not, by using the following command:

```
$ ibstat
```

Please look at the "State" field for the status of the port being used. To bring a port to "Active" status, on any node in the same InfiniBand subnet, execute the following command:

```
# opensm -o 1
```

Please note that you need superuser privilege for this command. This command invokes the InfiniBand subnet manager (OpenSM) and asks it to sweep the subnet once and make all ports "Active". OpenSM is usually installed in `/usr/local/bin`.

7.2.5 Couldn't modify SRQ limit

This means that your HCA doesn't support the `ibv_modify_srq` feature. Please upgrade the firmware version and OpenFabrics Gen2 libraries on your cluster. You can obtain the latest Mellanox firmware images from this webpage.

Even after updating your firmware and OpenFabrics Gen2 libraries, if you continue to experience this problem, please edit `make.mvapich.gcc` and replace `-DMEMORY_SCALE` with `-DADAPTIVE_RDMA_FAST_PATH`. After making this change you need to re-build the MVAPICH library. Note that you should first try to update your firmware and OpenFabrics Gen2 libraries before taking this measure.

If you believe that your HCA supports this feature and yet you are experiencing this problem, please contact the MVAPICH community by posting a note to mvapich-discuss@cse.ohio-state.edu mailing list.

7.2.6 Got completion with error code 12

The error code 12 indicates that the InfiniBand HCA has given up after attempting to send the packet after several tries. This can be caused by either loose or faulty cables. Please check the InfiniBand connectivity of your cluster. Additionally, you may check the error rates at the respective HCAs using:

```
$ ibchecknet
```

This utility (usually installed in `/usr/local/bin`) sweeps the InfiniBand subnet and reports ports that are OK or if they have errors. You may try to quiesce the entire cluster and bring it up after an InfiniBand switch reboot.

7.2.7 Hang with VIADEV_USE_LMC=1

The VIADEV_USE_LMC parameter allows the usage of multiple paths for multi-core and multi-way SMP systems, set up the subnet manager 9.1.6. The subnet manager allows different routing engines to be used (Min-Hop routing algorithm by default). We have noticed hangs using this parameter with Up/Down routing algorithm of OpenSM. There are two ways to fix this problem:

- Disable the VIADEV_USE_LMC. This can be done in the following manner.

  ```
  # mpirun_rsh -np 2 n0 n1 VIADEV_USE_LMC=0 ./prog
  ```

- Use the Min-Hop Algorithm. This can be done by invoking opensm with the min-hop algorithm. Please use the following command, which provides an LMC value of 4, and makes sure that the LIDs are re-assigned using the -r option.

  ```
  # opensm -o -l4 -r
  ```

7.3 Troubleshooting with MVAPICH/VAPI

7.3.1 Cannot Open HCA

The above error reports that the InfiniBand Adapter is not ready for communication. Make sure that the drivers are up. This can be done by executing:

```
$ locate libvapi
```

This command gives the path at which drivers are setup. Additionally, you may try to use the command `vstat` to check the availability of HCAs.

```
$ vstat
```

7.3.2 Cannot include vapi.h

This error is generated during compilation, if the correct path to the InfiniBand library installation is not given.

Please setup the environment variable MTHOME as

```
$ export MTHOME=/usr/local/ibgd/driver/infinihost
```

If the problem persists, please contact your system administrator or reinstall your copy of IBGD. You can get IBGD from Mellanox website.

7.3.3 Program aborts with `VAPI_RETRY_EXEC_ERROR`

This error usually indicates that all InfiniBand links the MPI application is trying to use are not in the `PORT_ACTIVE` state. Please make sure that all ports show `PORT_ACTIVE` with the VAPI utility `vstat`. If you are using Multi-Rail support, please keep in mind that all ports of all adapters you are using need to show `PORT_ACTIVE`.

7.3.4 ld:multiple definitions of symbol _calloc error on MacOS

Please make sure that the environmental variable `"MAC_OSX"` is set before your configuration. If you use manual configuration and not `mvapich.make.macosx`, you must configure MVAPICH in the following way:

```
$ export MAC_OSX=yes; ./configure; make; make install
```

If you encounter this problem of compiling your own applications, like given below, it is likely that you have explicitly included `"-lm"`. You should remove that.

```
   "ld: multiple definitions of symbol _calloc
/usr/lib/libm.dylib(malloc.So) definition of _calloc
/tmp/mvapich-0.9.5/mvapich/lib/libmpich.a(dreg-g5.o)
definition of _calloc in section (__TEXT,__text)
ld: multiple definitions of symbol _free
/usr/lib/libm.dylib(malloc.So) definition of _free
/tmp/mvapich-0.9.5/mvapich/lib/libmpich.a(dreg-g5.o)
definition of _free in section (__TEXT,__text) "
```

7.3.5 No Fortran interface on the MacOS platform

To enable Fortran support, you would need to install the IBM compiler located at (there is a 60-day free trial version) available from IBM.

Once you unpack the tar ball, you can customize and use `make.mvapich.vapi` to compile and install the package or manually configure, compile and install the package.

7.4 Troubleshooting with MVAPICH/UDAPL

7.4.1 Cannot Open IA

If you configure MVAPICH 0.9.9 with uDAPL and see this error, you need to check whether you have specified the correct uDAPL service provider. If you have specified the uDAPL provider but still see this error, you need to check whether the specified network is working or not.

In addition, please check the contents of the file `/etc/dat.conf`. It should contain the name of the IA e.g. ib0. A typical entry would look like the following:

```
ib0 u1.2 nonthreadsafe default /usr/lib/libdapl.so ri.1.1 ‘‘mthca0
1’’ ‘‘’’
```

7.4.2 DAT Insufficient Resource

If you configure MVAPICH 0.9.9 with uDAPL and see this error, you need to reduce the value of the environmental variable `RDMA_DEFAULT_MAX_WQE` depending on the underlying network.

7.4.3 Cannot find libdat.so

If you get the error: “error while loading shared libraries, libdat.so”, the location of the dat shared library is incorrect. You need to find the correct path of `libdat.so` and export `LD_LIBRARY_PATH` to this correct location. For example:

```
$ export LD_LIBRARY_PATH=/path/to/libdat.so:$LD_LIBRARY_PATH
```

```
$ mpirun_rsh -np 2 n0 n1 ./a.out
```

7.5 Troubleshooting with MVAPICH/QLogic InfiniPath

7.5.1 Low Bandwidth

Incorrect settings of MTRR mapping may result in achieving a low bandwidth with InfiniPath hardware. To alleviate this situation, BIOS settings for MTRR mapping may be edited to "Discrete". For further details, please refer to the InfiniPath User Guide.

7.5.2 Cannot find -lpsm_infinipath

Variable IBHOME_LIB in make.mvapich.psm file does not point to correct location. IBHOME_LIB should point to the directory containing the InfiniPath device libraries. By default they are installed in /usr/lib or /usr/lib64.

7.5.3 Mandatory variables not set

IBHOME, PREFIX, CC, F77 are mandatory variable required by the installation script and must be set in the file make.mvapich.psm. IBHOME - directory which contains the InfiniPath header file include directory. By default InfiniPath header file include directory is in /usr. PREFIX - directory where MVAPICH should be installed. CC - C compiler. Typically set to gcc. F77 - fortran compiler. Typically set to g77.

7.5.4 Can't open /dev/ipath, Network Down

This probably means that the ports are not configured to be a part of the InfiniBand subnet and thus are not "Active". You can check whether the port is active or not, by using the following command on that node:

```
$ ipath_control -i
```

Please look at the "Status" field for the status of the port being used. To bring a port to "Active" status, on any node in the same InfiniBand subnet, execute the following command:

```
# opensm -o
```

Please note that you may need superuser privilege for this command. This command invokes the InfiniBand subnet manager (OpenSM) and asks it to sweep the subnet once and make all ports "Active". OpenSM is usually installed in `/usr/local/bin`. You may also look at the file /sys/bus/pci/drivers/ib_ipath/status_str to verify that the InfiniPath software is loaded correctly. For details, please refer to InfiniPath user guide, download able from www.qlogic.com.

7.5.5 No ports available on /dev/ipath

This is a limitation of InfiniPath Release 2.0. By default, QHT7140 allows a maximum of eight node programs per node and a maximum of four node programs per node with the QLE7140. To overcome this, please consult your InfiniPath support provider.

8 Tuning and Scalability Features for Large Clusters

MVAPICH supports many different parameters for tuning and extracting the best performance for a wide range of platforms and applications. These parameters can be either compile time parameters or run time parameters. Please refer to section 9 for a complete description of all the parameters. In this section we classify these parameters depending on what you are tuning for and provide guidelines on how to use them.

8.1 Network Point-to-point Tuning

In MVAPICH 0.9.9, we introduce a new highly scalable mode of operation which consumes very less memory. It can lead to significant reduction in the memory footprint of MVAPICH.

To enable this mode, please include `-DMEMORY_SCALE` in your `make.mvapich.gcc` (it is included by default). Once you have enabled the scalable memory mode in MVAPICH, there are three aspects by which you can customize the memory usage and performance ratio according to the needs of your cluster.

8.1.1 Shared Receive Queue (SRQ) Tuning

The main environmental parameters controlling the behavior of the Shared Receive Queue design are:

- VIADEV_SRQ_SIZE (9.4.2)
- VIADEV_SRQ_LIMIT (9.4.3)
- VIADEV_VBUF_POOL_SIZE (9.2.7)

VIADEV_SRQ_SIZE is the maximum size of the Shared Receive Queue. You may increase this to value 1000 if the application requires very large number of processors (4K and beyond). VIADEV_SRQ_LIMIT defines the low watermark for the flow control handler. This can be reduced if your aim is to reduce the number of interrupts.

VIADEV_VBUF_POOL_SIZE is a fixed number of pool of `vbufs`. These vbufs can be shared among all different connections depending on the communication needs of each connection. You may want to increase this number for large scale clusters (4K and beyond).

8.1.2 On-Demand Connection Tuning

The major environmental variables controlling the behavior of the connection management in MVAPICH are:

- VIADEV_CM_RECV_BUFFERS (9.8.1)
- VIADEV_CM_MAX_SPIN_COUNT (9.8.2)
- VIADEV_CM_TIMEOUT (9.8.3)

VIADEV_CM_RECV_BUFFERS is the number of buffers used by the connection manager to establish new connections. These buffers are very small (around 20 bytes) and they are shared for all InfiniBand connections, so this value may be increased to 8192 for large clusters to avoid retries in case of packet drops.

VIADEV_CM_MAX_SPIN_COUNT is the number of times the connection manager polls for new incoming connections. This may be increased to reduce the interrupt overhead when lot of incoming connections are started at the same time.

VIADEV_CM_TIMEOUT is the timeout value associated with connection request messages on the UD channel. Decreasing this may lead to faster retries, but at the cost of generating duplicate messages. Similarly increasing this may lead to slower retries but lesser chance of duplicate messages.

8.1.3 Adaptive RDMA Tuning

MVAPICH implements a dynamic allocation and utilization of the RDMA mechanism for short messages. It can lead to significant reduction in memory footprint of MVAPICH.

There are two environmental parameters:

- VIADEV_ADAPTIVE_RDMA_LIMIT (9.2.17)
- VIADEV_ADAPTIVE_RDMA_THRESHOLD (9.2.18)

These two parameters control the behavior of this dynamic scheme.
VIADEV_ADAPTIVE_RDMA_LIMIT controls the maximum number of processes for which the "fast" RDMA buffers are allocated. For very large scale clusters, it is suggested to set this value to -1, which means RDMA buffers will be allocated for $log(n)$ number of connections (where n is the number of processes in the application).
VIADEV_ADAPTIVE_RDMA_THRESHOLD is the number of messages exchanged per connection before RDMA buffers are allocated for that connection. For very large scale clusters, it is suggested that this value be increased so that only very frequently communicating connections allocate RDMA buffers.

In addition, the following parameters are also important in tuning the memory requirement: VIADEV_VBUF_TOTAL_SIZE (9.2.2) and VIADEV_NUM_RDMA_BUFFER (9.2.1).

The product of VBUF_TOTAL_SIZE and VIADEV_NUM_RDMA_BUFFER generally is a measure of the amount of memory registered for eager message passing. These buffers are not shared across connections.

To provide the best performance (latency/bandwidth) to memory ratio, we have decided on a set of default values for these parameters. These parameters are often dependent on the execution platform. To use preset values for small, medium and large clusters (1-64, 64-256, 256-. . .), please use VIADEV_CLUSTER_SIZE (9.9.1) as either SMALL, MEDIUM or LARGE, respectively.

8.2 Shared Memory Point-to-point Tuning

MVAPICH uses shared memory communication channel to achieve high-performance message passing among processes that are on the same physical node. The two main parameters which are used for tuning shared memory performance for small messages are VIADEV_SMPI_LENGTH_QUEUE (9.5.2) and VIADEV_SMP_EAGER_SIZE (9.5.1). The two main parameters which are used for tuning shared memory performance for large messages are SMP_SEND_BUF_SIZE(9.5.3) and VIADEV_SMP_NUM_SEND_BUFFER (9.5.4).

VIADEV_SMPI_LENGTH_QUEUE is the size of the shared memory buffer which is used to store outstanding small and control messages. VIADEV_SMP_EAGER_SIZE defines the switch point from Eager protocol to Rendezvous protocol.

Messages larger than VIADEV_SMP_EAGER_SIZE are packetized and sent out in a pipelined manner. SMP_SEND_BUF_SIZE is the packet size, i.e. the send buffer size. VIADEV_SMP_NUM_SEND_BUFFER is the number of send buffers. Shared memory communication can be disabled at run time by the parameter VIADEV_USE_SHARED_MEM(9.3.5).

Performance of some applications is sensitive to the rank distribution according to their communication pattern. It is advisable that processes that communicate most use the shared memory path, since it offers lower latencies compared to the network path. To adjust the process rank distribution, please refer Section 5.2 to decide which distribution "cyclic" or "block" suits the communication pattern of your application. In particular, we have found that when using "block" distribution, the performance of HPL (Linpack) is better.

9 MVAPICH Parameters

9.1 InfiniBand HCA and Network Parameters

9.1.1 VIADEV_DEVICE

- Class: Run time
- Default: First IB device found on the system

Name of the InfiniBand device. e.g. `mthca0`, `mthca1` or `ehca0` (for IBM ehca).

9.1.2 VIADEV_DEFAULT_PORT

- Class: Run time
- Default: First IB port found on the system

The default port on the InfiniBand device to be used for communication.

9.1.3 VIADEV_MAX_PORTS

- Class: Run time
- Default: 2

This variables allows to change the maximum number of ports per adapter which are supported.

9.1.4 VIADEV_USE_MULTIHCA

- Class: Run time
- Default: 0

This variable allows a user to bind processes on a node to ports attached to different HCAs on a node. It allows an efficient utilization of HCA ports in a round-robin fashion. VIADEV_MULTIHCA is an alias for this variable for backward compatibility. However, if VIADEV_USE_MULTIHCA is defined, value of VIADEV_MULTIHCA will be overwritten.

9.1.5 VIADEV_USE_MULTIPORT

- Class: Run time
- Default: 0

This variable allows a user to bind processes on a node to ports attached to different HCAs on a node. It allows an efficient utilization of HCA ports in a round-robin fashion. VIADEV_MULTIPORT is an alias for this variable for backward compatibility. However, if VIADEV_USE_MULTIPORT is defined, value of VIADEV_MULTIPORT will be overwritten.

9.1.6 VIADEV_USE_LMC

- Class: Run time
- Default: 0

This variable allows the usage of multiple paths between end nodes for multi-core/multi-way SMP systems. The path selection is on the basis of source and destination ranks.

9.1.7 VIADEV_DEFAULT_MTU

- Class: Run time
- Default: MTU1024

The internal MTU used for IB. This parameter should be a string instead of an integer. Valid values are: `MTU256`, `MTU512`, `MTU1024`, `MTU2048`, `MTU4096`.

9.2 Memory Usage and Performance Control Parameters

9.2.1 VIADEV_NUM_RDMA_BUFFER

- Class: Run time
- Default: Architecture dependent (32 for IA-32)

The number of RDMA buffers used for the RDMA fast path. This *fast path* is used to reduce latency and overhead of small data and control messages. This value is effective only when macro RDMA_FAST_PATH or ADAPTIVE_RDMA_FAST_PATH is defined.

9.2.2 VIADEV_VBUF_TOTAL_SIZE

- Class: Run time
- Default: Architecture dependent (6 KB for EM64T)

This macro defines the size of each `vbuf`.

Different presets for this value are available for different sizes of clusters VIADEV_CLUSTER_SIZE = (SMALL, MEDIUM, LARGE, AUTO).

9.2.3 VIADEV_RNDV_PROTOCOL

- Class: Run time
- Default: RPUT

This parameter chooses the underlying Rendezvous protocol Other options are RGET (allows for more overlap) and R3 (allows to send large messages without using registration cache)

9.2.4 VIADEV_RENDEZVOUS_THRESHOLD

- Class: Run time
- Default: Architecture dependent (12KB for IA-32)

This specifies the switch point between eager and rendezvous protocol in MVAPICH.

9.2.5 VIADEV_MAX_RDMA_SIZE

- Class: Run time
- Default: 1048576

Maximum size of an RDMA put message (RPUT) in the rendezvous protocol. Note that this variable should be set in bytes.

9.2.6 VIADEV_R3_NOCACHE_THRESHOLD

- Class: Run time
- Default: 1048576

This is the message size (in bytes) which will be sent using the R3 mode if the registration cache is turned off, i.e. VIADEV_USE_DREG_CACHE=0

9.2.7 VIADEV_VBUF_POOL_SIZE

- Class: Run time
- Default: 5000

The number of vbufs in the initial pool. This pool is shared among all the connections.

9.2.8 VIADEV_VBUF_SECONDARY_POOL_SIZE

- Class: Run time
- Default: 500

The number of vbufs allocated each time when the global pool is running out in the initial pool. This is also shared among all the connections.

9.2.9 VIADEV_USE_DREG_CACHE

- Class: Run time
- Default: 1

This indicates whether registration cache is to be used or not. The registration cache speeds up zero copy operations if user memory is re-used many times.

9.2.10 VIADEV_NDREG_ENTRIES

- Class: Run time
- Default: 1000

This defines the total number of buffers that can be stored in the registration cache. A larger value will lead to more infrequent lazy de-registration.

9.2.11 VIADEV_DREG_CACHE_LIMIT

- Class: Run time
- Default: No limit

This sets a limit on the number of pages kept registered by the registration cache. If you set it to 0, that implies no limits on the number of pages registered.

9.2.12 VIADEV_VBUF_MAX

- Class: Run time
- Default: -1 (No limit)

Max (total) number of VBUFs to allocate after which the process terminates with a fatal error. -1 means no limit.

9.2.13 VIADEV_ON_DEMAND_THRESHOLD

- Class: Run time
- Default: 32

Number of processes beyond which on-demand connection management will be used.

9.2.14 VIADEV_MAX_INLINE_SIZE

- Class: Run time
- Default: 128

Maximum size of a message (in bytes) that may be sent INLINE with message descriptor Lowering this increases message latency, but can lower memory requirements. Also see VIADEV_NO_INLINE_THRESHOLD, which will override this value in some cases.

9.2.15 VIADEV_NO_INLINE_THRESHOLD

- Class: Run time
- Default: 256

This parameter automatically changes the VIADEV_MAX_INLINE_SIZE after the number of connections exceeds VIADEV_NO_INLINE_THRESHOLD. Behavior is slightly different depending on whether on-demand connection setup is used:

- If the number of processes in a job is less than VIADEV_ON_DEMAND_THRESHOLD, then the maximum inline size for all connections is automatically set to zero to save memory.
- If the number of processes is greater than VIADEV_ON_DEMAND_THRESHOLD, then the first VIADEV_NO_INLINE_THRESHOLD number of connections per process have an inline size of VIADEV_MAX_INLINE_SIZE and all subsequent connections have a maximum inline size of zero.

9.2.16 VIADEV_USE_BLOCKING

- Class: Run time
- Default: 0

Use blocking mode progress, instead of polling. This allows MPI to yield CPU to other processes if there are no more incoming messages.

9.2.17 VIADEV_ADAPTIVE_RDMA_LIMIT

- Class: Run Time
- Default: Number of processes in application

This is the maximum number of RDMA paths that will be established in the entire MPI application. Passing it a value -1 implies that at most $log(n)$ number of paths will be established. Where n is the number of processes in the MPI application.

9.2.18 VIADEV_ADAPTIVE_RDMA_THRESHOLD

- Class: Run Time
- Default: 10

This is the number of messages exchanged per connection after which the RDMA path is established.

9.2.19 VIADEV_ADAPTIVE_ENABLE_LIMIT

- Class: Run Time
- Default: 32

Default value: Number of processes (np) in application If the number of jobs exceeds this limit, adaptive flow will be enabled. To enable adaptive flow for any number of jobs define: VIADEV_ADAPTIVE_ENABLE_LIMIT=0

9.2.20 VIADEV_SQ_SIZE

- Class: Run time
- Default: 40

To control the number of allowable outstanding send operations to the device.

9.3 Send/Receive Control Parameters

9.3.1 VIADEV_CREDIT_PRESERVE

- Class: Run time
- Default: 100

This parameter records the number of credits per connection that will be preserved for non-data, control packets. If SRQ is not used, this default is 10.

9.3.2 VIADEV_CREDIT_NOTIFY_THRESHOLD

- Class: Run time
- Default: 5

Flow control information is usually sent via piggybacking with other messages. This parameter is used, along with VIADEV_DYNAMIC_CREDIT_THRESHOLD, to determine when to send explicit flow control update messages.

9.3.3 VIADEV_DYNAMIC_CREDIT_THRESHOLD

- Class: Run time
- Default: 10

Flow control information is usually sent via piggybacking with other messages. These two parameters are used to determine when to send explicit flow control update messages.

9.3.4 VIADEV_INITIAL_PREPOST_DEPTH

- Class: Run time
- Default: 5

This defines the initial number of pre-posted receive buffers for each connection. If communication happen for a particular connection, the number of buffers will be increased to VIADEV_PREPOST_DEPTH.

9.3.5 VIADEV_USE_SHARED_MEM

- Class: Run time
- Default: 1

When _SMP_ is defined, shared memory communication can be disabled by setting VIADEV_USE_SHARED_MEM=0.

9.3.6 VIADEV_PROGRESS_THRESHOLD

- Class: Run time
- Default: 1

This value determines if additional MPI progress engine calls are made when making send operations. If there are this number or more queued send operations then progress is attempted.

9.3.7 VIADEV_USE_COALESCE

- Class: Run time
- Default: 1

This setting turns on (1) or off (0) the coalescing of messages. Leaving feature on can help applications that make many consecutive send operations to the same host.

9.3.8 VIADEV_COALESCE_THRESHOLD_SQ

- Class: Run time
- Default: 4

If there are more than this number of small messages outstanding to a another task, messages will be coalesced until one of the previous sends completes.

9.3.9 VIADEV_COALESCE_THRESHOLD_SIZE

- Class: Run time
- Default: VIADEV_VBUF_TOTAL_SIZE

Attempt to coalesce messages under this size. If this number is greater than VIADEV_VBUF_TOTAL_SIZE, then it is set to VIADEV_VBUF_TOTAL_SIZE. This has no effect if message coalescing is turned off.

9.4 SRQ (Shared Receive Queue) Control Parameters

9.4.1 VIADEV_USE_SRQ

- Class: Run Time
- Default: 1

Indicates whether Shared Receive Queue is to be used or not. Users are strongly encouraged to use this as long as the InfiniBand software/hardware supports this feature.

9.4.2 VIADEV_SRQ_SIZE

- Class: Run Time
- Default: 512

This is the maximum number of work requests allowed on the Shared Receive Queue.

9.4.3 VIADEV_SRQ_LIMIT

- Class: Run Time
- Default: 30

This is the low watermark limit for the Shared Receive Queue. If the number of available work entries on the SRQ drops below this limit, the flow control will be activated.

9.4.4 VIADEV_MAX_R3_OUST_SEND

- Class: Run Time
- Default: 32

This is the maximum number of R3 packets which are outstanding when using Shared Receive Queues.

9.4.5 VIADEV_SRQ_ZERO_POST_MAX

- Class: Run Time
- Default: 1

Maximum number of unsuccessful SRQ posts that an async thread can make before going to sleep.

9.4.6 VIADEV_MAX_R3_PENDING_DATA

- Class: Run Time
- Default: 524288

This is the maximum amount of R3 data that is sent out un-acked

9.5 Shared Memory Control Parameters

9.5.1 VIADEV_SMP_EAGERSIZE

- Class: Run time
- Default: Architecture dependent (32KB for EM64T)

This has no effect if macro _SMP_ is not defined. It defines the switch point from Eager protocol to Rendezvous protocol for intra-node communication. If macro _SMP_RNDV_ is defined, then for messages larger than VIADEV_SMP_EAGERSIZE, SMP Rendezvous protocol is used. Note that this variable should be set in KBytes.

9.5.2 VIADEV_SMPI_LENGTH_QUEUE

- Class: Run time
- Default: Architecture dependent (128KB for EM64T)

This has no effect if macro _SMP_ is not defined. It defines the size of shared buffer between every two processes on the same node for transferring messages smaller than or equal to VIADEV_SMP_EAGERSIZE. Note that this variable should be set in KBytes.

9.5.3 SMP_SEND_BUF_SIZE

- Class: Compile time
- Default: Architecture dependent (8KB for EM64T)

This has no effect if macro _SMP_ is not defined. It defines the packet size when sending intra-node messages larger than VIADEV_SMP_EAGERSIZE. Note that this variable should be set in Bytes.

9.5.4 VIADEV_SMP_NUM_SEND_BUFFER

- Class: Run time
- Default: Architecture dependent (128 for EM64T)

This has no effect if macro _SMP_ is not defined. It defines the number of internal send buffers for sending intra-node messages larger than VIADEV_SMP_EAGERSIZE.

9.5.5 VIADEV_USE_AFFINITY

- Class: Run time
- Default value: 1

Enable CPU affinity by setting VIADEV_USE_AFFINITY=1 or disable it by setting VIADEV_USE_AFFINITY=0. VIADEV_USE_AFFINITY does not take effect when _AFFINITY_ is not defined.

9.6 Multi-Rail Usage Parameters

9.6.1 STRIPING_THRESHOLD

- Class: Run time
- Default: VIADEV_RENDEZVOUS_THRESHOLD

For a class of messages, a user may want to use Rendezvous protocol and not stripe the data across multiple ports/adapters. For messages of size equal and above this value, the data is striped across multiple paths. This value should at least be equal to the VIADEV_RENDEZVOUS_THRESHOLD. The value of STRIPING_THRESHOLD is currently equal to VIADEV_RENDEZVOUS_THRESHOLD. For optimal performance, this value may need to be changed depending upon the multi-rail setup (i.e. the number of ports and number of adapters) in the system.

9.6.2 NUM_QP_PER_PORT

- Class: Run time
- Default: 2

This parameter indicates number of queue pairs per port to be used for communication on an end node. This parameter has no effect if Multi-Rail configuration is not enabled.

9.6.3 NUM_PORTS

- Class: Run time
- Default: 2

This parameter indicates number of ports to be used for communication per adapter on an end node. This parameter has no effect if Multi-Rail configuration is not enabled.

9.6.4 NUM_HCAS

- Class: Run time
- Default: 1

This parameter indicates number of adapters to be used for communication on an end node. This parameter has no effect if Multi-Rail configuration is not enabled.

9.6.5 SM_SCHEDULING

- Class: Run time
- Default: ROUND_ROBIN

To control the scheduling policy being used for small messages for Multi-Rail device. Valid policies are `USE_FIRST` (only use the first sub channel), `ROUND_ROBIN` (use subchannels in a round-robin manner) and `PROCESS_BINDING` (bind processes to a specific port of the HCAs). This parameter is only valid for the OpenFabrics/Gen2 Multi-Rail device.

9.6.6 LM_SCHEDULING

- Class: Run time
- Default: STRIPE_BLOCKING

To control the scheduling policy being used for large messages for Multi-Rail device. Valid policies are `ROUND_ROBIN` (use subchannels in a round-robin manner), `WEIGHTED_STRIPING` (weight subchannels according to their link rates), `EVEN_STRIPING` (use equal weights for all subchannels), `STRIPE_BLOCKING` (stripe messages based on whether they are blocking or non-blocking MPI messages), `ADAPTIVE_STRIPING` (adaptively change the weights based on network congestion) and `PROCESS_BINDING` (bind processes to a specific port of the HCAs).

9.7 Run time parameters for Collectives

9.7.1 VIADEV_USE_RDMA_BARRIER

- Class: Run time
- Default: 0

To enable RDMA based Barrier, set this to 1.

9.7.2 VIADEV_USE_RDMA_ALLTOALL

- Class: Run time
- Default: 0

To enable RDMA based Alltoall, set this to 1.

9.7.3 VIADEV_USE_RDMA_ALLGATHER

- Class: Run time
- Default: 0

To enable RDMA based Allgather, set this to 1.

9.7.4 VIADEV_USE_SHMEM_COLL

- Class: Run time
- Default: 1

To disable SHMEM based collectives, set this to 0.

9.7.5 VIADEV_USE_SHMEM_BARRIER

- Class: Run time
- Default: 1

To disable SHMEM based Barrier, set this to 0.

9.7.6 VIADEV_USE_SHMEM_ALLREDUCE

- Class: Run time
- Default: 1

To disable SHMEM based Allreduce, set this to 0.

9.7.7 VIADEV_USE_SHMEM_REDUCE

- Class: Run time
- Default: 1

To disable SHMEM based Reduce, set this to 0.

9.7.8 VIADEV_MAX_SHMEM_COLL_COMM

- Class: Run time
- Default: 4

This parameter allows to configure the number of communicators using shared memory collectives.

9.7.9 VIADEV_SHMEM_COLL_MAX_MSG_SIZE

- Class: Run time
- Default: $1 \ll 20$

This parameter allows the maximum message to be tuned for the shared memory collectives.

9.7.10 VIADEV_SHMEM_COLL_REDUCE_THRESHOLD

- Class: Run time
- Default: $1 \ll 10$

The shmem reduce is taken for messages less than this threshold. This threshold can be tuned appropriately but should be less than that of 9.7.9 above.

9.7.11 VIADEV_SHMEM_COLL_ALLREDUCE_THRESHOLD

- Class: Run time
- Default: $1 \ll 15$

The shmem allreduce is taken for messages less than this threshold. This threshold can be tuned appropriately but should be less than that of 9.7.9 above.

9.8 CM Control Parameters

9.8.1 VIADEV_CM_RECV_BUFFERS

- Class: Run time
- Default: 1024

To control the number of receive buffers dedicated to UD based connection manager. Each buffer is only several tens of bytes.

9.8.2 VIADEV_CM_MAX_SPIN_COUNT

- Class: Run time
- Default: 5000

9.8.3 VIADEV_CM_TIMEOUT

- Class: Run time
- Default: 500 ms

To control the timeout value for UD messages.

9.9 Other Parameters

9.9.1 VIADEV_CLUSTER_SIZE

- Class: Run time
- Default: Small

This controls the preset values for vbuf size, number of RDMA buffers and Rendezvous threshold for various cluster sizes. It can be set to “SMALL” (1-64), “MEDIUM” (64-256) and “LARGE” (256 and beyond). In addition, there is an “AUTO” option which will automatically set the appropriate parameters based on number of processes in the MPI application.

9.9.2 VIADEV_PREPOST_DEPTH

- Class: Run time
- Default: 64

This defines the number of buffers pre-posted for each connection to handle send/receive operations.

9.9.3 VIADEV_MAX_SPIN_COUNT

- Class: Run time
- Default: 1000

This parameter is only effective when blocking mode progress is used. This parameter indicates the number of polls made by MVAPICH before yielding the CPU to other applications.

9.9.4 VIADEV_PT2PT_FAILOVER

- Class: Run time
- Default: 256 MB

This is the memory size of RDMA-based implementations for Alltoall and Allgather after which the default point-to-point mechanism is used instead of RDMA.

9.9.5 DAPL_PROVIDER

- Class: Run time
- Default: ib0

This is to specify the underlying uDAPL library that the user would like to use if MVAPICH is built with uDAPL.